Overview

• Introduction to Mi-V™

• Why we need a compliance and verification platform

• Cross platform verification and compliance testing of Mi-V processor cores

• Applying formal model checking to core verification
Mi-V RISC-V Ecosystem

- A continually expanding, comprehensive suite of tools and design resources to fully support RISC-V designs
- Aims to increase adoption of RISC-V ISA and Microchip's soft CPU product family
- Supports development using Microchip’s soft-CPUs and RISC-V SoC FPGAs
Mi-V Compliance and Verification

• We’d like to be able to integrate and use open-source cores.

  • In some cases, these have unknown verification and unknown compliance to the RISC-V specification(s).

  • Whilst the base cores are chosen with some (varying) level of verification expected, we may need to make changes/additions, so these must be verified.

• We’d like all cores to be compliant.

  • We would like all cores to be interoperable with all tools and all other cores, therefore they must be compliant.

  • Microchip contributes to the compliance working group.

  • This is an opportunity to test out the compliance suite across multiple platforms and improve it as part of the ongoing development of RISC-V.

  • Propose methodology for user and privileged “super-compliance.”
Mi-V Compliance and Verification

• RISC-V VIP
  • Compliance
  • Torture
  • Simulation environment

• Mi-V Compliance Suite
  • Compliance
  • Hardware/emulation/simulation environment

• Benchmarks
RISC-V VIP

- Uses an AHB interface
- Can pass selections of test hex files to the core and compare the resulting memory dump with reference signatures
• **Can be configured to run**
  • Compliance tests
  • Torture tests

• **Compliance tests**
  • RISC-V compliance GitHub
  • Support for I, M and C extensions

• **Torture tests**
  • Generated from Clifford Wolf’s scripts
  • Support for I, M & C extensions
  • 1000 tests for 8 configurations: 8000 tests
  • Coverage
Mi-V RISC-V Compliance

- Make each test a function, and call it from a main
- Use global defines to enable/disable tests
Mi-V RISC-V Compliance

- Create a buffer and print the content at the end of each test to minimize the size
RISC-V Compliance Emulation Platform
RISC-V Compliance Simulation Platform
Simulation (and emulation) is insufficient for processor verification.

- Processors are complex (even simple ones!).
- Processors have many corner cases, in fact a lot of the logic is to manage such cases.
- Can simulation ever hit all the permutations, both spatially and temporally?
- Do we know what to hit, i.e. is the coverage sufficient?

Formal methods are good for processor verification.

- Very good for finding corner case issues quickly.
- Processors are well specified and easily constrained (have well-defined interfaces).
  - RISC-V is particularly well specified!
- Allow aggressive design.
- Super-compliance.
• Compare RTL implementation to a formal model
  • Approach used is similar to other approaches checking architectural transformation of state such as ARM® ISA-Formal, RISC-V Formal (Clifford Wolf).

• Mi-V Formal differs significantly due to the targets and goals
  • Not assuming designs are our own, therefore requires a portable, black-box approach
  • Cannot rely on most state being visible

• Mi-V Formal checks equivalence to an architectural model by comparing the program counter of retired instructions
Known Knowns, Known Unknowns...

• Some model states are unknown, some are architecturally undefined, some have a range of allowable behavior – how do we check equivalency with an implementation?
  • Initialize everything in both model and implementation – hard to do and inefficient
  • Use a model that deals with it

• Created SystemVerilog ISA model that allows known/unknown state (and propagation of it) to be explicitly defined with “known value” logic.
  • Avoids the need for any “directed” constraints to avoid undesirable behavior that would normally be discounted as “bad software.” Only constrained by the architecture definition (not over-constrained).
  • “Known value” propagation allows us to aggressively abstract memories, resets and arithmetic operations, etc., by only caring about the concrete values in a small subset of cases.
Known Knowns, Known Unknowns...

- “Known value” logic model
  - Created 4-state logic model.
  - Defined ISA model based on this logic. In SystemVerilog this means a know-value datatype (tuple of value, known/unknown-state) and functions for all logic/arithmetic functions.
  - Example:
    - A = \{0101, 1111\} would be a value of 0101 with all bits known.
    - If B = \{0111, 0110\} then A&B = \{0101, 0110\}, and A+B = \{1010, 0000\}.

- Propagation of failures
  - Example:
    - CSR has incorrect access privilege in implementation where implementation incorrectly allows access.
      - csrrw x0, mtvec, x0 <model pc = \{0,F\}, impl pc =0>
      - csrxxx x?, <offending_csr>, x0 <model pc = \{4,F\}, impl pc =4>
      - any instruction <model pc = \{0,F\}, impl pc =8>
• Mi-V Formal has been focused on design bring-up (bug-hunting) and on privileged architecture compliance on machine-mode cores.
• Highly efficient, highly effective!
• Bugs found include:
  • Incorrect CSR accesses, privileges and bit writability
  • Incorrect trap behavior
  • Incorrect exception return behavior
  • Getting stuck in debug mode
  • Pipeline interlock errors
  • Decode errors
  • Fetch and load/store ordering
  • …
Summary

- For development and release of Mi-V RISC-V cores, Microchip has employed a suite of traditional techniques and combined them with formal model checking techniques to enhance quality and ensure compliance.

- Traditional techniques have been run on multiple platforms, utilizing open-source compliance, random-instruction set simulation and benchmarks, with added coverage measurement.

- Formal techniques have been developed to provide better/faster verification of new cores as well as privileged ISA compliance.

- Mi-V core RTL and compliance/verification suite, are available as part of Microchip’s Libero software and open-sourced on GitHub at https://github.com/RISCV-on-Microsemi-FPGA.